Experiences With OpenVault,
A Prototype Implementation of a Standards Effort

Curtis Anderson
Silicon Graphics, Inc.
2011 N. Shoreline Blvd, M/S 802
Mountain View, CA 94043
curtis@sgi.com
Voice: +1 650 933-1193
FAX: +1 650 933-3542

1.0 Introduction

The Storage Systems Standards Working Group (http://www.ssswg.org) is developing new removable media management standards—defining a set of testable standards that can be used by end users and industry to build multi-vendor storage environments. These standards will allow end users to combine the most appropriate pieces from each vendor to best meet their needs.

Silicon Graphics has provided engineering resources to implement a prototype of the proposed design to validate the architectural approach. That prototype is called OpenVault. This paper describes that effort and what we have learned so far.

2.0 Architectural Requirements

The architecture for OpenVault was chosen to meet requirements for library devices and for drives. A drive can be viewed as a device that applications know about directly and interact with. A library device, on the other hand, can be completely hidden from the application by a storage management system. The storage management system's only job is to move named cartridges into compatible drives so that applications can use them. In the future, a Mover will probably be added to this architecture to hide drive-specific issues from the application as well. The major requirements for a media management system are discussed in the following sections.

2.1 Separation of Device-Dependent and Device-Independent Software

Traditional removable storage management systems build library-device control directly application software. This linkage severely limits the user's ability to choose library hardware that is not supported by the applications. It also imposes a large burden on the application vendor in doing regression testing of each supported library with each new release of the application (or the underlying OS).

Traditional systems also build drive dependencies into the application. The current IEEE project is not tackling that issue yet, as we believe that it requires a Mover to isolate the application from drive idiosyncrasies.

2.2 Sharing of Hardware Resources

To effectively manage a library device, it is easiest to dedicate that device to one storage management application, the approach taken in almost all current management applications.

For low-usage or occasional-use storage applications, dedicating a library device to the application is generally expensive. Obtaining larger library devices, shared between applications, is both more flexible and more cost effective.

Some of today’s libraries can be “partitioned” in a fixed manner between several storage management applications. Unfortunately, this usually requires that the drives be partitioned as well. Dedicating drives to applications is not only expensive, it is inefficient in terms of available system bandwidth.

Thus, the ability to use all of the available (compatible) drives for currently high priority tasks would be of great benefit to the end-user.

2.3 Cross-Platform Portability

Application vendors desire standard interfaces to storage management services on all the platforms that they support. This allows them to lower the cost of supporting each platform. To meet this desire, a storage management system should be portable to multiple platforms while providing the same interface on all platforms.

2.4 Centralized Storage System Management

System administrators desire one central place to manage their storage environment; if they have several large servers with tape drives, they would like to manage all the cartridge pools on all of their servers at the same time according to consistent policies.

Sites must be able to strike their own balance between local autonomy and central control, which implies that both local autonomy and central control must be possible.

2.5 Run Time Plug-And-Play

The separation of device-dependent code from device-independent code implies a standard interface between the two portions. Since standard interfaces allow components to be built by different parties, the system must be able to support run-time plug-and-play of components with components coming from multiple sources.

In a large and diverse market, a central registration authority for components is not practical, so a non-centrally-controlled system is required.

2.6 Network Security

To operate in distributed systems, systems must be able to protect themselves from attacks via the network. To allow export from the United States, pure encryption of the channel is not an option.

3.0 Architectural Overview

At its most abstract, the media management environment is composed of five major components, each of which is most likely a separately running process or set of processes:

- Client applications,
- Library Managers (LMs),
- Drive Managers (DMs),
- The Media Manager (MM), and
- A Catalog (part of the MM).

We will describe each of these pieces and then present a rationale for each design. We will define these five pieces indirectly by defining the communications protocol between them.

3.1 ASCII Languages and TCP/IP Sockets

The interprocess communication mechanism from clients, LMs, and DMs to the MM is plain ASCII text passed through TCP/IP sockets. Well-defined languages are parsed to recover the meaning of the messages being passed. The syntax and semantics of each language is designed for the pair of components that it is connecting.

The currently defined media-management languages are asynchronous: Many commands can be outstanding at any time and responses are not required to come back in the order that the commands were submitted. The languages are also fully bi-directional in that each end of the socket has the same level of asynchrony available to it, allowing each end to have commands outstanding with the other end at the same time.

3.2 The HELLO Protocol and Version Negotiation

All the network connections in this architecture form a star topology with the MM at the center of the star. Each of those connections is initiated by the piece outside the MM—the MM never initiates connections.

When an application, LM, or DM first contacts the MM they participate in a protocol that announces to the MM details about the connecting process. Those details include which language and which versions of that language the process is able to speak. From the language name the MM can determine what type of process is connecting, e.g., client, LM, or DM.

If there are no problems, the MM responds with a positive acknowledgment and the language version it has chosen to use during this session. Otherwise, the MM will respond with an error message and will close the connection. Examples of problems include an unknown language name or a non-supported language version.

3.3 Security and Message Integrity Codes

The initial contact from the client includes the name of the client and a name for this instance of that client. The MM uses that information to look up a shared, secret password for that client and instance.

If a non-null key exists, then a per-session unique encryption key is generated. The key is used to generate a digital signature for each command passed through the TCP/IP socket. The digital signature follows the clear text of the command and allows the receiver to determine if the message has been modified in transit.

If no key can be found then only the clear text of the command passes through the socket.

3.4 The MM and the Catalog

The Media Manager is the central coordinator for the environment. It makes use of the services offered by the attached LMs and DMs to accomplish the tasks requested by applications.

An internal component of the MM, called the Catalog, persistently stores the characteristics of the objects being managed and processes arbitrarily complex user-specified queries against that set of data. The results of those queries are used to identify the objects that the application wants to operate on. The catalog objects being operated on include cartridges, drives, libraries, applications, cartridge groups, and drive groups.

3.5 Applications and MMP

Client applications may be full hierarchical storage management systems or simple Perl scripts accessing removable media cartridges. They communicate with the MM via a language called the Media Management Protocol (MMP). MMP has multiple privilege levels to support both privileged and unprivileged clients.

The major capabilities that an unprivileged client can make use of in MMP are:

- Obtaining ownership of a cartridge from a scratch pool and releasing it again,
- Mounting and unmounting cartridges that the application owns,
- Storing and retrieving application defined metadata on owned cartridges, and
- Viewing information about the current composition and status of the system.

A privileged client application adds at least the following abilities:

- modifying the configuration and composition of the system,
- managing queued tasks, and
- providing operator interaction support for LMs and DMs if required.

MMP is composed of many separate commands, but there are also clauses in the language that are common to most of those commands. The most important of these is the “match” clause which determines the objects on which the command should operate.

The match clause is an arbitrarily complex algebraic expression that defines the criteria controlling inclusion into the set of catalog objects operated on by the command. It is a declaration of constraints that all solutions must satisfy. In theory, the expression is evaluated once for each possible combination of objects in the catalog. For each such evaluation that returns “true”, that combination of objects will be included in the set of objects that will be operated on by the command. In practice, techniques more efficient than brute force can be used.

For example, if a client application wants to mount a cartridge such that it will have greater than 9 MB/sec of nominal read bandwidth from the drive, the application can express that constraint in almost exactly those terms. Given the known cartridges, the currently connected Library and Drive Managers, and the characteristics of the devices they control, the MM must find a solution that satisfies the application’s constraint.

3.6 The LM and LMP

A Library Manager is a specialized interface process that implements high-level commands by understanding the particular library device’s operational characteristics and the available device-specific commands. LMs communicate with the MM through the Library Management Protocol (LMP) which defines an ASCII interface to an abstract Library.

The LMP is not just a command set, it defines a relationship between an LM and the MM where they each have responsibilities to the other. The LM’s primary responsibilities are to implement the commands that come down from the MM and to keep the MM updated on any changes in the state or configuration of the library device. The LM is the authoritative source for all information about the library and is required to tell the MM whenever anything changes that would invalidate the MM’s cached copy of that information.

The LM describes the library to the MM in a stylized structure incorporating three basic elements: slots, bays, and drives. A slot and a drive are just what you would expect. A bay is best described by visualizing a group of Storage Technology silos connected by pass-through ports. Each silo in that picture is a “bay”.. Thus, a bay is a collection of slots and drives that form a locality group in terms of access time. When the LM reports the slots to the MM, it includes the external label (a bar code, for example) along with the type of cartridge (DLT, 3480, etc.).

3.7 The DM and DMP

A Drive Manager is a specialized interface process that implements high-level commands by understanding the particular drive’s operational characteristics and the available device-specific commands. DMs are similar to LMs except that they control drives instead of libraries. Their language is the Drive Manager Protocol (DMP) which defines an ASCII interface to an abstract drive; the DMP defines a relationship between a DM and the MM where each one has responsibilities to the other.

The DMs primary responsibilities are to implement the commands from the MM and to keep the MM updated on changes in the state or configuration of the drive. The DM is the authoritative source for all information about the drive and tells the MM whenever changes invalidate the MM’s cached copy of that information.

Drives usually support multiple modes (e.g., compression or not), and varying levels of backward compatibility. For example, a DLT7000 drive can read but not write a DLT2000 cartridge. The DM describes the drive to the MM in a stylized structure that lists the supported combinations of various characteristics. Those characteristics include the type of cartridge, the format of the bits on the media, and a set of “capability strings”.

Examples of cartridge types are “DLT” and “3480”, while examples of bit recording formats are “3480” and “3490”. A “capability string” is an uninterpreted token that the DM provides to the MM. Applications also provide such tokens to the MM during a mount operation. This allows the MM to rendezvous the application’s desires with the abilities of the drive. Examples of capability strings include “rewind” and “compression”.

The description that the DM sends to the MM contains an unordered set of named combinations. Each combination is defined by the values of “cartridge type”, “bit format”, and a particular set of “capability string” tokens. The MM treats each combination as a separate “mode” that the drive supports.

When evaluating a “match” clause for an MMP command, if the MM finds a match between the known characteristics of a candidate cartridge (cartridge type and bit format) and a match between the “capability strings” required by the application with those offered by the drive, then the MM will conclude that that drive can be used to access that cartridge in the way that the application desires.

4.0 Architectural Rationale

Telling customers that they must buy an additional tape robot when they add a new robot-aware application to their system is not popular, yet that is where the SCSI-attached robot market is today. If the value of that application is not enough to justify an entire robot or a high-performance drive, then the site has to make do with a smaller robot or with a less capable drive. To grow their market, the storage industry must break that linkage and share robots and drives between multiple applications.

The requirement to separate device-dependent from device-independent software led to the concept of Library Managers and Drive Managers as separate entities. This allows device-specific code to deal with the idiosyncrasies of each device while isolating that code from the core mechanisms of the system. Device-specific code also allows implementors to maximize the performance and reliability of each device.

Defining the relationship between a DM or an LM and the MM was a delicate balancing act which we will describe in this section. In designing the LMP and the DMP, we chose to migrate complexity from the LM and the DM to the MM to reduce the implementation effort for all of the LM and DM writers.

4.1 ASCII Languages and TCP/IP Sockets

The requirement for portable, run-time plug-and-play support led us to choose simple ASCII languages through TCP/IP sockets as the interprocess communication mechanism. This is the most portable solution since all major operating systems support TCP/IP and ASCII strings. Binary RPC mechanisms are less portable and make asynchronous commands more difficult to implement.

Defining the linkages between components rather than defining the components themselves allows implementors to create new components and to plug them into running systems.

4.2 The HELLO Protocol and Version Negotiation

The “hello” protocol supports system evolution; new versions of existing languages and entirely new languages can be defined, with the MM negotiating which language and which version to use at connection setup time. This allows both gradual upgrades of components and mixed-version environments.

The syntax of the protocol flowing through the socket after the successful completion of the hello protocol is dependent on the language and language version that were negotiated during the hello protocol. Thus, a binary language, rather than an ASCII one, can be defined if desired.

4.3 Security and Message Integrity Codes

Message integrity codes (MICs) were chosen over pure encryption of the command stream so that products can be exported from the United States.

Since the text of the command is in the clear just preceding the integrity code, an eavesdropper can see the command traffic flowing back and forth but cannot change it. Since no end-user data flows through the command channels, the MIC technique is a good compromise.

4.4 The MM and the Catalog

It is likely that many LMs and DMs will be written to handle the variety of current and future devices. On the other hand, we expect only a few MMs to be written from scratch. It is possible that a single, standard MM will be developed, ported, and enhanced over time. Thus, we decided to move complexity from the LMs and DMs (and storage-accessing applications) into the MM. Part of that complexity was providing a persistent storage mechanism that the LMs, DMs, and applications could use to manage system information, simplifying the task of developing LMs and DMs.

Another area of complexity was in the Catalog's support of the MMP's "match" clause. Defining the match semantics and designing efficient algorithms was difficult. There are many implicit relationships between the objects stored by the Catalog, and yet the results from a mount command and commands that report the contents of the Catalog must be consistent. For example, a "volume" is an application-accessible object that exists on a "cartridge". If a client sends a "match" clause saying that the volume color must be blue and the cartridge color must be red, the result should include only blue volumes that exist on red cartridges, not all red volumes plus all blue cartridges. The rationale is that the only cartridge that is relevant to a volume is the cartridge containing the volume. Although complex, the "match" clause provides a general mechanism that is able to handle current and future requests through one interface.

4.5 The Application and MMP

To gain initial acceptance of this architecture, existing storage management applications must begin to adopt it. This implies that we need to reduce the amount of application rework needed to begin using the MM architecture. The design of the MMP allows an application to start with basic functionality, like mounting a named cartridge into a named drive, and grow into more advanced operations such as using the "match" clause over application-specific metadata stored on objects in the MM system.

4.6 The LM and LMP

The protocol between the LM and the MM must provide a common look and feel for all robots and yet be able to take advantage of the unique capabilities of each robot design. We chose to make the LM an active partner of the MM, rather than a slave, to give the LM the ability to react to robot changes asynchronously and to initiate state changes in the MM as a result.

To be general, the MM must not be dependent on any particular hardware model. The slots and bays that the LM describes to the MM, for example, are defined solely by the LM and are named with strings. The MM only compares those strings for equality with other strings. Thus, the MM can handle any library that is defined in terms of slots and bays, and the LM is free to map those constructs onto reality in any convenient way.

The MM must also present a view of the physical hardware to administrative applications and humans. Thus, the strings that the LM gives to the MM to name slots and bays must be human readable, and the human must be able to map them onto physical features of the library device.

4.7 The DM and DMP

The DM needs to support the abstract drive that the DMP language defines while making use of the unique capabilities of each physical drive. Like the LM, we chose to make the DM an active partner to the MM rather than a slave.

To allow the MM to decide if a drive can be used to access a given cartridge, the DM must present the capabilities of the drive to the MM in a standard way. For example, the DMP must be able to represent all of the possible combinations of cartridge types and bit formats. This will require, at least, a standard set of token definitions.

4.8 The Lack of a Mover (So Far)

The current architecture supports local access to drives via the native operating system access methods (e.g., /dev nodes on UNIX) and leaves a hole for later addition of a Mover component. We believe this is an acceptable limitation at this time but acknowledge that a true Mover is required in the long term. Without a Mover, the MM will not be able to protect internal labels from malicious applications on platforms that do not provide that protection through the operating system. The MM will be able to control access to drives, however, by carefully managing either the permissions on the drive access handle (e.g., the /dev/mt node) or by creating and deleting the drive handle on demand. The particular technique used to control access to the drive is DM implementation specific.

5.0 Future Directions

There are several capabilities that we would like to add to the system. The architecture described here has the power and flexibility required to include these capabilities without a major redesign.

5.1 Addition of a Mover

There are two major advantages to having a Mover plus a few disadvantages. A Mover in the data path can intercept any repositioning commands and protect an internal label from modification by unauthorized clients, even if the operating system provide no protection for the label.

A Mover will also provide access to drives that are not physically connected to the system on which the application is running. This will allow sites to centralize removable media resources and to share devices that would otherwise be too expensive for local applications to justify.

The major challenge with a Mover component is that it must have very low overhead during data transfers. The Mover interface must be able to faithfully reproduce the semantics of the drive it is connected to, or applications must be modified to use the Mover semantics. Neither of these options is trivial.

A Mover in the MMS architecture, like the DM, will be device- and OS-specific much as a DM is device- and OS-specific. This allows the Mover to accommodate drive idiosyncrasies and allows device-specific optimizations, but requires an interface that will be common to all Movers.

5.2 Network Attached Storage

The rapid evolution of network-attached storage devices in the market today makes it difficult to predict future directions. The basic concepts of removable media will remain the same, but the details of managing such environments will change. The goal is to manage network-attached devices through the same interfaces used to manage host-attached devices.

5.3 Multiple-Simultaneous-Access Media Types

The popular removable media devices today are inherently exclusive-access devices. For example, tape drives do not support multiple applications accessing different portions of the tape at the same time. Newer types of removable media will allow simultaneous accesses.

DVDROM and the new magneto-optical disks are examples. Truly simultaneous access to a single volume might be limited to read-only access, but simultaneous read/write access to different partitions on a given cartridge should be possible.

5.4 Time Based Scheduling

With the convergence of computers and television, the concept of time-based scheduling will be needed in storage management systems. With TV stations using robotic storage to manage commercials, a natural MMS request will be to “mount this tape for access at 5:28pm”.. The storage management system is the only software in a position to know all the demands being placed on the removable media system, and therefore must control the scheduling of drive access.

6.0 Lessons Learned So Far

The concept of using LMs and DMs and abstract languages to connect them to a central management engine has worked out very well. The idea of providing a declarative, constraint-based, arbitrary query mechanism has proved to be powerful and general. It is not yet clear what the correct balance should be between the most intuitive relationships between the objects represented in the MM and the most convenient relationships.